CSSE 220 Day 22

Threads and Animation

Check out *ThreadsIntro* project from SVN

Multithreaded programs

- Often we want our program to do multiple (semi) independent tasks at the same time
- Each thread of execution can be assigned to a different processor, or one processor can simulate simultaneous execution through "time slices" (each typically a large fraction of a millisecond)

Time → Slices	1	2	3	4	5	6	7	8	9	10	11	12	13	14
running thread 1	■	■		■				■		■			■	■
running thread 2			■		■	■	■		■		■	■		

Q1-2

Why use Threads?

- Animation: runs while still allowing user interaction
- A server (such as a web server) communicates with multiple clients
- Allow a slow activity to occur in the background
  - Example: While a game is loading its (large) data files, another thread might display an interesting animation to the player or ask the user for relevant information
- Animate multiple objects, e.g.
  - Each Ball in BallWorlds
  - The timers in the soon-to-be-seen **CounterThreads** example
- In general, allow separate objects to "do their thing" separately

A Java Program's Threads

- There are always two default threads:
  - The one that starts in *main*
  - The one that handles events

You can create others

- What can you do with a Thread?
  - Construct it
  - Start it
  - Suspend it

    ```
    Thread.sleep(numberOfMilliseconds);
    ```
  - Interrupt it, perhaps to cause it to halt

Q3-4

The Emperor's New Threads

- How to construct and run a new thread
  1. Define a new class that implements the **Runnable** interface
     - Runnable has one method: `public void run();`
  2. Place the code for the threaded task in the `run` method:
     ```
     class MyRunnable implements Runnable {
        public void run () {
           // task statements go here; presumably a loop
        }
     }
     ```
  3. Create an object of this class:
     ```
     Runnable r = new MyRunnable();
     ```
  4. Construct a Thread object from this Runnable object:
     ```
     Thread t = new Thread(r);
     ```
  5. Call the **start** method to start the thread:
     ```
     t.start();
     ```

Note: a common pattern is to have the Runnable construct and start its own Thread in its constructor:

```
new Thread(this).start();
```

Q5

Threads examples (in your SVN repos.)

Open Eclipse and enter the SVN repository perspective. Then:

1. Refresh your individual repository
2. Checkout the *ThreadsIntro* project you see there

We will run and study some of its subprojects:

- **Greetings** –simple threads, different wait times
- **AnimatedBall** – move balls, stop with click
- **CounterThreads** – multiple independent counters
- **CounterThreadsRadioButtons** – same as above, but with radio buttons

The remaining are more advanced than we will use in this course, dealing with race conditions and synchronization. Detailed descriptions are in *Big Java* Chapter 20

- **BankAccount**
- **SelectionSorter**

Simple example (1) – greetings Output

One thread prints the **Hello** messages; the other Thread prints the **Goodbye** messages.

Each thread sleeps for a random amount of time after printing each line.

Try it yourself!

```
Thu Jan 03 16:09:36 EST 2008 Hello, World!
Thu Jan 03 16:09:36 EST 2008 Goodbye, World!
Thu Jan 03 16:09:36 EST 2008 Hello, World!
Thu Jan 03 16:09:36 EST 2008 Goodbye, World!
Thu Jan 03 16:09:36 EST 2008 Goodbye, World!
Thu Jan 03 16:09:36 EST 2008 Hello, World!
Thu Jan 03 16:09:37 EST 2008 Goodbye, World!
Thu Jan 03 16:09:37 EST 2008 Hello, World!
Thu Jan 03 16:09:38 EST 2008 Hello, World!
Thu Jan 03 16:09:38 EST 2008 Goodbye, World!
Thu Jan 03 16:09:38 EST 2008 Goodbye, World!
Thu Jan 03 16:09:38 EST 2008 Hello, World!
Thu Jan 03 16:09:39 EST 2008 Goodbye, World!
Thu Jan 03 16:09:39 EST 2008 Goodbye, World!
Thu Jan 03 16:09:39 EST 2008 Goodbye, World!
Thu Jan 03 16:09:39 EST 2008 Hello, World!
Thu Jan 03 16:09:39 EST 2008 Hello, World!
Thu Jan 03 16:09:39 EST 2008 Goodbye, World!
Thu Jan 03 16:09:40 EST 2008 Hello, World!
Thu Jan 03 16:09:40 EST 2008 Goodbye, World!
. . . .
```

This example was adapted from Cay Horstmann's *Big Java 3ed*, Chapter 20

Simple example(2) – GreetingThreadTester

```java
public class GreetingThreadTester{

  public static void main(String[] args){

    // Create the two Runnable objects
    GreetingRunnable r1 = new GreetingRunnable("Hello, World!");
    GreetingRunnable r2 = new GreetingRunnable("Goodbye, World!");

    // Create the threads from the Runnable objects
    Thread t1 = new Thread(r1);
    Thread t2 = new Thread(r2);

    // Start the threads running.
    t1.start();
    t2.start();
   }
}
```

We do not call **`run()`** directly.
Instead we call **`start()`**, which sets up the thread environment and then calls **run()** for us.

Simple example(3) – a Runnable class

```
import java.util.Date;

public class GreetingRunnable implements Runnable {

   private String greeting;
   private static final int REPETITIONS = 15;
   private static final int DELAY = 1000;

   public GreetingRunnable(String aGreeting) {
      this.greeting = aGreeting;
   }

   public void run() {
      try {
         for (int i = 1; i <= GreetingRunnable.REPETITIONS; i++){
            Date now = new Date();
            System.out.println(now + " " + this.greeting);
            Thread.sleep(
               (int) (GreetingRunnable.DELAY * Math.random()));
         }
      } catch (InterruptedException exception) {
               ; // Do nothing, just continue running
      }
   }
}
```

If a thread is interrupted while it is sleeping, an **InterruptedException** is thrown.

Q6

Ball Animation

- A simplified version of the way BallWorlds does animation
- When balls are created, they are given position, velocity, and color
- Our `run()` method tells each of the balls to move, then redraws them
- Clicking the mouse turns movement off/on
- Demonstrate the program

Set up the frame

```
public class AnimatedBallViewer {

   static final int FRAME_WIDTH = 600;
   static final int FRAME_HEIGHT = 500;

   public static void main(String[] args){
      JFrame frame = new JFrame();

      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setTitle("BallAnimation");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      AnimatedBallComponent component = new AnimatedBallComponent();
      frame.add(component);

      frame.setVisible(true);
      new Thread(component).start();
   }
}
```

This class has all of the usual stuff, plus this last line of code that starts the animation.

The Ball class

```
class Ball {
   private double centerX, centerY, velX, velY;
   private Ellipse2D.Double ellipse;
   private Color color;
   private static final double radius = 15;

   public Ball(double cx, double cy, double vx, double vy, Color c){
      this.centerX = cx;
      this.centerY = cy;
      this.velX = vx;
      this.velY = vy;
      this.color = c;
      this.ellipse = new Ellipse2D.Double (
          this.centerX-radius, this.centerY-radius,
          2*radius, 2*radius);
   }

   public void fill (Graphics2D g2) {
      g2.setColor(this.color);
      g2.fill(ellipse);
   }

   public void move (){
      this.ellipse.x += this.velX;
      this.ellipse.y += this.velY;
   }
}
```

Everything here should look familiar, similar to code that you wrote for BallWorlds.

AnimatedBallComponent: Instance Variables and Constructor

```
public class AnimatedBallComponent extends JComponent
                   implements Runnable, MouseListener {

   private ArrayList<Ball> balls = new ArrayList<Ball>();
   private boolean moving = true;
   public static final long DELAY = 30;
   public static final int ITERATIONS = 300;

   public AnimatedBallComponent() {
      super();
      balls.add(new Ball(40, 50, 8, 5, Color.BLUE));
      balls.add(new Ball(500, 400, -3, -6, Color.RED));
      balls.add(new Ball(30, 300, 4, -3, Color.GREEN));
      this.addMouseListener(this);
   }
```

Again, there should be no surprises here!

AnimatedBallComponent: run, paintComponent, mousePressed

One could let this loop run forever [while (true) { … }] but we chose here to make sure that it ends

```
public void run() {
    for (int i=0; i<ITERATIONS; i++) {
        if (moving){
            for (Ball b:balls)
                b.move();
            this.repaint();
        }
        try {
            Thread.sleep(DELAY);
        } catch (InterruptedException e) {}
    }
}

  public void paintComponent(Graphics g){
    Graphics2D g2 = (Graphics2D)g;
    for (Ball b:balls)
        b.fill(g2);
  }

  public void mousePressed (MouseEvent arg0) {
    moving = !moving;
  }
```

Each time through the loop (if moving), tell each ball to move, then repaint

Sleep for a while

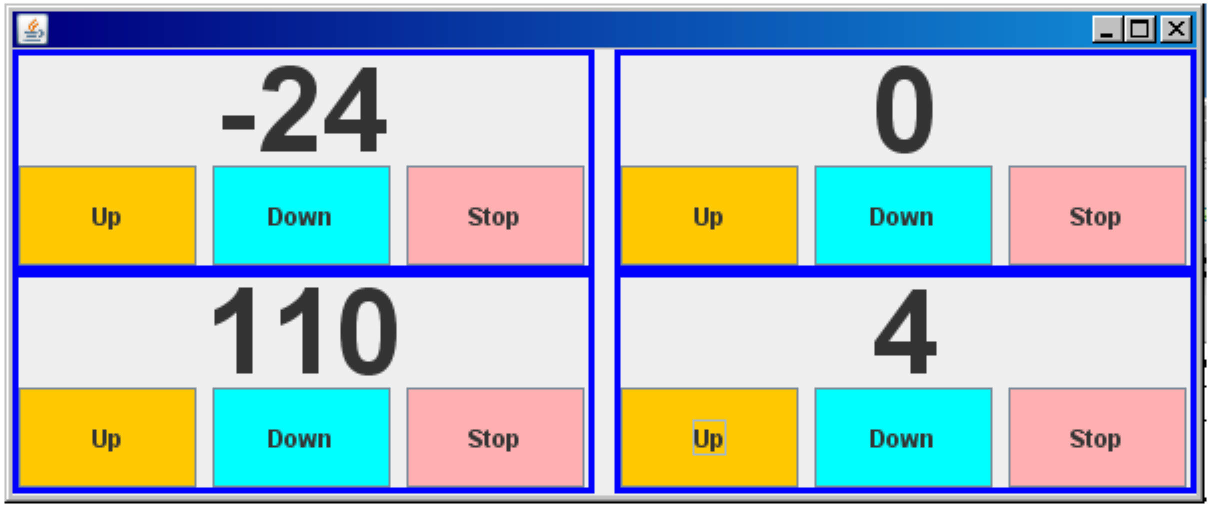
Draw each ball

Toggle "moving" when the mouse is pressed

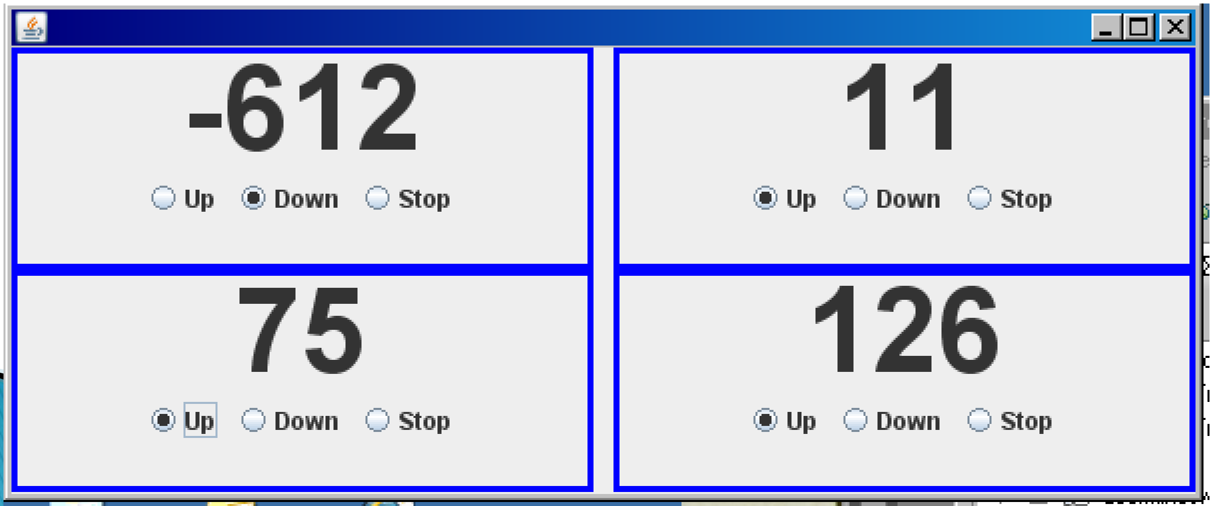
Q7

Another animation: CounterThreads

- With regular buttons



With radio buttons



Run it.

How many threads does this application appear to have?

CounterThreads setup

```
public class CounterThreads {

  public static void main (String []args) {
    JFrame win = new JFrame();
    Container c = win.getContentPane();
    win.setSize(600, 250);
    c.setLayout(new GridLayout(2, 2, 10, 0));
    c.add(new CounterPane(200));
    c.add(new CounterPane(500));
    c.add(new CounterPane(50)); // this one will count fast!
    c.add(new CounterPane(1000));

    win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    win.setVisible(true);
   }
}
```

Same old stuff!

CounterPane Basics

```
class CounterPane extends JComponent implements Runnable {

  private int delay;    // sleep time before changing counter
  private int direction = 0; //  current increment of counter
  private JLabel display = new JLabel("0");

  // Constants to define counting directions:
  private static final int COUNT_UP    =  1; // Declaring these
  private static final int COUNT_DOWN   = -1; // constants avoids
  private static final int COUNT_STILL  =  0; // "magic numbers"

  private static final int BORDER_WIDTH =  3;
  private static final int FONT_SIZE    = 60;
```

CounterPane Constructor

```
public CounterPane(int delay) {

    JButton upButton   = new JButton("Up");     // Note that these do
    JButton downButton = new JButton("Down");   // NOT have to be fields
    JButton stopButton = new JButton("Stop");   // of this class.

    this.delay = delay; // milliseconds to sleep

    this.setLayout(new GridLayout(2, 1, 5, 5));
       // top row for display, bottom for buttons.

    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout(1, 3, 8, 1));
    display.setHorizontalAlignment(SwingConstants.CENTER);
    display.setFont(new Font(null, Font.BOLD, FONT_SIZE));
       // make the number display big!

    this.add(display);
    this.add(buttonPanel);
    this.setBorder(BorderFactory.createLineBorder(Color.blue,
                                                  BORDER_WIDTH));
    // Any Swing component can have a border.
    this.addButton(buttonPanel, upButton,   Color.orange, COUNT_UP);
    this.addButton(buttonPanel, downButton, Color.cyan,   COUNT_DOWN);
    this.addButton(buttonPanel, stopButton, Color.pink,   COUNT_STILL);

    Thread t = new Thread(this);
    t.start();
```

Put a simple border around the panel. There are also more complex border styles that you can use.

A lot of the repetitive work is done by the calls to **addButton()**.

CounterPane's addButton method

```
// Adds a control button to the panel, and creates an
// ActionListener that sets the count direction.
private void addButton(Container container,
                       JButton button,
                       Color color,
                       final int dir) {
  container.add(button);
  button.setBackground(color);
  button.addActionListener(new ActionListener () {
     public void actionPerformed(ActionEvent e) {
       this.direction = dir;
     }
  });
}
```

JPanel is a subclass of Container

The value of **dir** will be 1, −1, or 0, to indicate counting up, down, or neither.

- The action listener added here is an anonymous inner class that implements ActionListener.
- Because it is an inner class, its method can access this CounterPane's **direction** instance variable and the addButton's *final* **dir** local variable.

Note that each button gets its own ActionListener class, created at runtime. This is Swing's "preferred way" of providing ActionListeners.

CounterPane's run method

- This method is short and simple, because direction is always the amount to be added to the counter (1, –1, or 0).

```
public void run() {
    try {
      do {
        Thread.sleep(delay);
        display.setText(Integer.parseInt(display.getText())
                        + direction  + "");
      } while (true);
    } catch (InterruptedException e) { }
  }
}
```

CounterThreads questions

- Look through the code, discussing it with your partner and/or lab assistants until you think you understand it all. Answer the following questions:

1. How does a CounterPane know whether to count up or down or stay the same?
2. When a counter is not changing, does its thread use less CPU time than one that is changing?
3. Would it be easy to add code to the *main* method that creates a SuperStop button, so that clicking this button stops all counters? Explain.

Answer: Yes. Have CounterPane respond to the SuperStop button; hence all instances of CounterPane would respond.

Q8–10

RadioButton version

```
public CounterPaneRadio(int delay) {

    JRadioButton upButton   = new JRadioButton("Up");
    JRadioButton downButton = new JRadioButton("Down");
    JRadioButton stopButton = new JRadioButton("Stop");

    ButtonGroup group = new ButtonGroup();
    group.add(upButton);
    group.add(downButton);
    group.add(stopButton);
    stopButton.setSelected(true);

...
```

And we remove the `Color` parameter from `addButton()`

Ending a thread

- A thread ends when its **run** method terminates.
- You can cause its **run** method to terminate in either of two ways:
  1. Via the Runnable
  2. Via the Thread itself

The next slides show the details of these.

Q11

Ending a thread via the Runnable

```
public class Foo implements Runnable {
    private boolean stopNow = false;

    public void run() {
        while (! stopNow) {
            // do your tasks
        }
     }

    public void stopRunning() {
        this.stopNow = true;
    }
}
```

If an object calls stopRunning, the thread stops soon thereafter. (How soon?)

Ending a thread via the Thread itself

```
public class FooBar {
    private Thread thread;

    public FooBar() {
        this.thread =
            new Thread(new Foo());
        this.thread.start();
    }

    public void stopRunning() {
        this.thread.interrupt();
    }
}
```

```
public class Foo implements Runnable {

    public void run() {
        while (true) {
            try {
                // do your tasks
            } catch (
                InterruptException e) {
              return;
            }
        }
    }
}
```

If an object calls stopRunning, the thread stops soon thereafter. (How soon?)